PROJECT TEAM 2 – CIS043
DEVELOPMENT WITH JAVA PROGRAMMING 2024

BLACK JACK

Mission College Team 2 Section 76296

Professor: Helen Sun

Nehan Armin

Safi Ebeid

Ian Smith

Gwen Vasquez

Virgilio Ravelo

Contents

Introduction:

Blackjack is one of the most popular casino card games, often played between a player and a dealer. The goal of the game is to get a hand value as close to 21 as possible, without exceeding it, while having a higher hand value than the dealer. The game is usually played with one or more standard decks of 52 cards.

Here's an introduction to the key elements of Blackjack:

1. The Basics of the Game:

Objective: The objective is to beat the dealer by having a hand value of 21 or less that is higher than the dealer's hand. If you exceed 21, you "bust" and lose the round.

To make out game unique, we have added a SPECIAL game mode along with a NORMAL game mode.

Card Values:

Number cards (2 through 10) are worth their face value.

In NORMAL mode, face cards have a value of 10.

Face cards Jack, Queen, and King will have 11,12,13, respectively in SPECIAL mode.

Aces can be either 1 or 11.

2. Game Setup:

Typically, Blackjack is played with 1 to 8 decks of cards, shuffled together.

The game is usually played with a dealer who represents the house, and 1 or more players.

In NORMAL mode, the player is dealt two cards, and the dealer is dealt one card.

In SPECIAL mode, the player starts with one card. Instead of drawing two cards at the beginning, players will draw one card and can choose to draw a second card. If the player chooses to draw a second card, then the bid will double.

3. Basic Gameplay:

Initial Deal: Each player receives one or two cards depending on the game mode, and the dealer receives one card

Player's Turn: Starting from the left, each player decides how to play their hand. The options are:

Hit: Take an additional card to try to improve the hand.

Stand: Keep the current hand and end the turn.

Double Down: Double the initial bet and receive exactly one more card.

Surrender: Allow players to surrender, meaning they forfeit all their bet and end the hand immediately.

Dealer's Turn: After all players have completed their hands, the dealer reveals the face-down card. The dealer must hit if their total is 16 or less and stand on 17 or higher.

Winning: Players win by having a higher hand than the dealer without busting (going over 21). If both the player and dealer have the same total, it's a "push," and the player's bet is returned.

4. Blackjack:

If a player's cards total up to 21, then it's a Blackjack and that player automatically wins.

5. Betting:

Players place their bets before any cards are dealt.

Features:

Core Concepts of Blackjack

Player: A person or an entity (like the dealer) who participates in the game.

Deck: The set of cards from which players draw during the game.

Card: Represents a single card with a suit (hearts, spades, diamonds, clubs) and a rank (2, 3, 4, ... 10, Jack, Queen, King, Ace).

Game System: The controller that coordinates the actions between the player, dealer, and the deck.

Game Flow:

1. **Deck Initialization**: The deck is created and shuffled.
2. **Card Dealing**:

   (NORMAL Mode): The player is dealt two cards.

   (SPECIAL Mode): The player is dealt one card, and they choose to pick up a second card

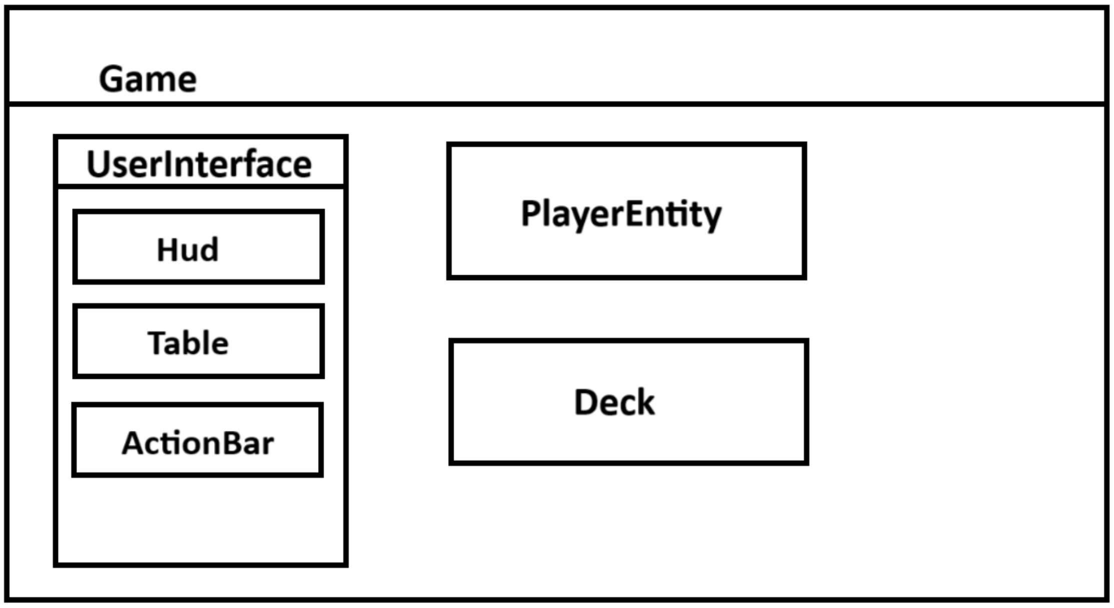
   The dealer is dealt one card.
3. **Player's Turn**: The player can choose to hit or stand.
4. **Dealer's Turn**: The dealer automatically draws cards until they reach a value of 17 or more.
5. **Winner Determination**: The game checks if the player or dealer has won based on the total value of each of their hands.

Object-Oriented Principles Used:

- **Encapsulation**: Classes like Card, Deck, Hand, and Player encapsulate data and behavior.
- **Inheritance**: UI classes inherit from JPanel and JFrame. Player and Dealer inherit from PlayerEntity.
- **Polymorphism**: The class PlayerEntity is able to accommodate more types of players than just the player and the dealer, due to its flexibility.
- **Abstraction**: The game logic is decoupled from the UI to allow for maximum extendibility and reusability. The UI is abstracted through the methods that it exposes to the Game logic.

Design:

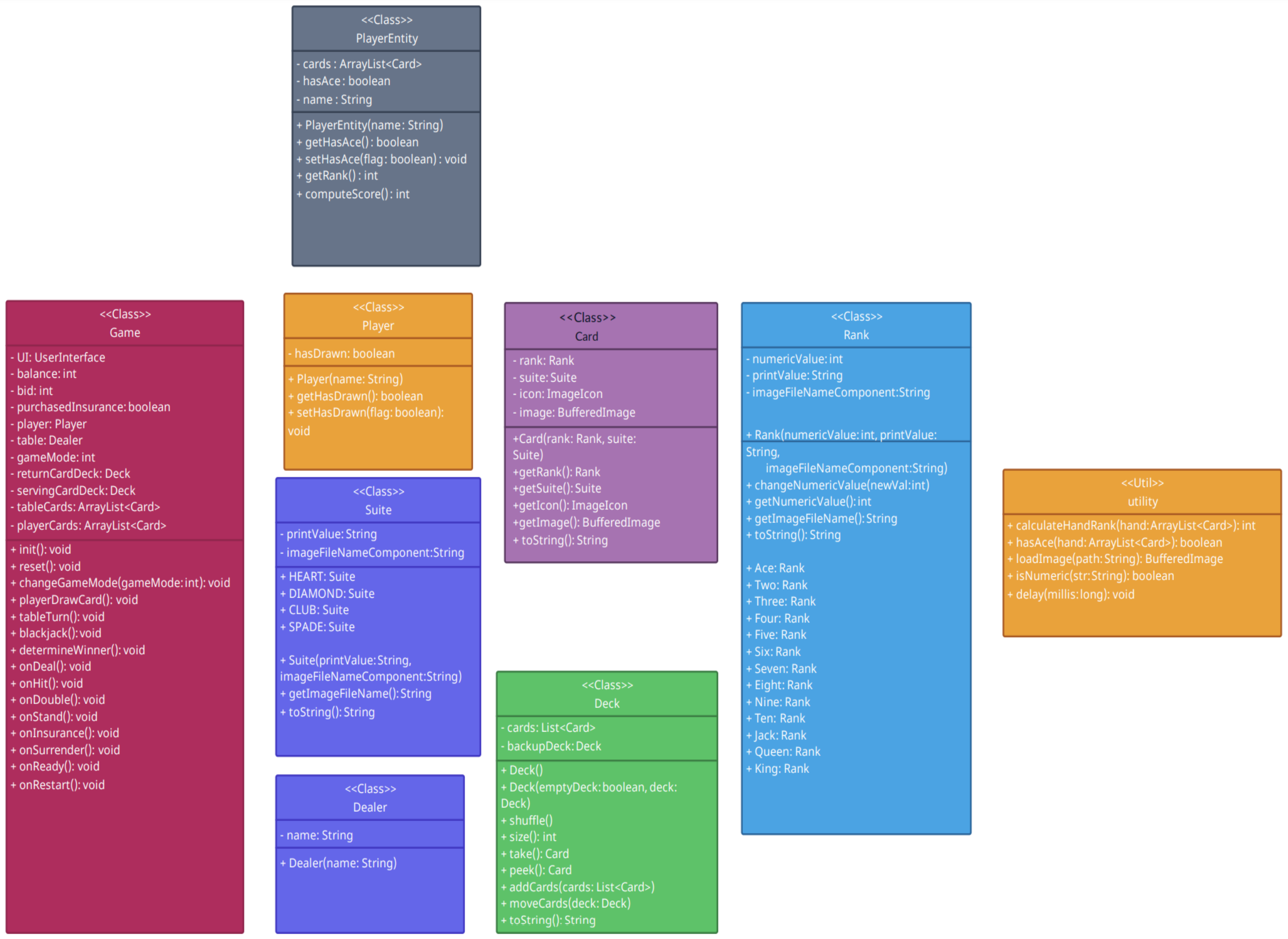
Object Structure Diagram:



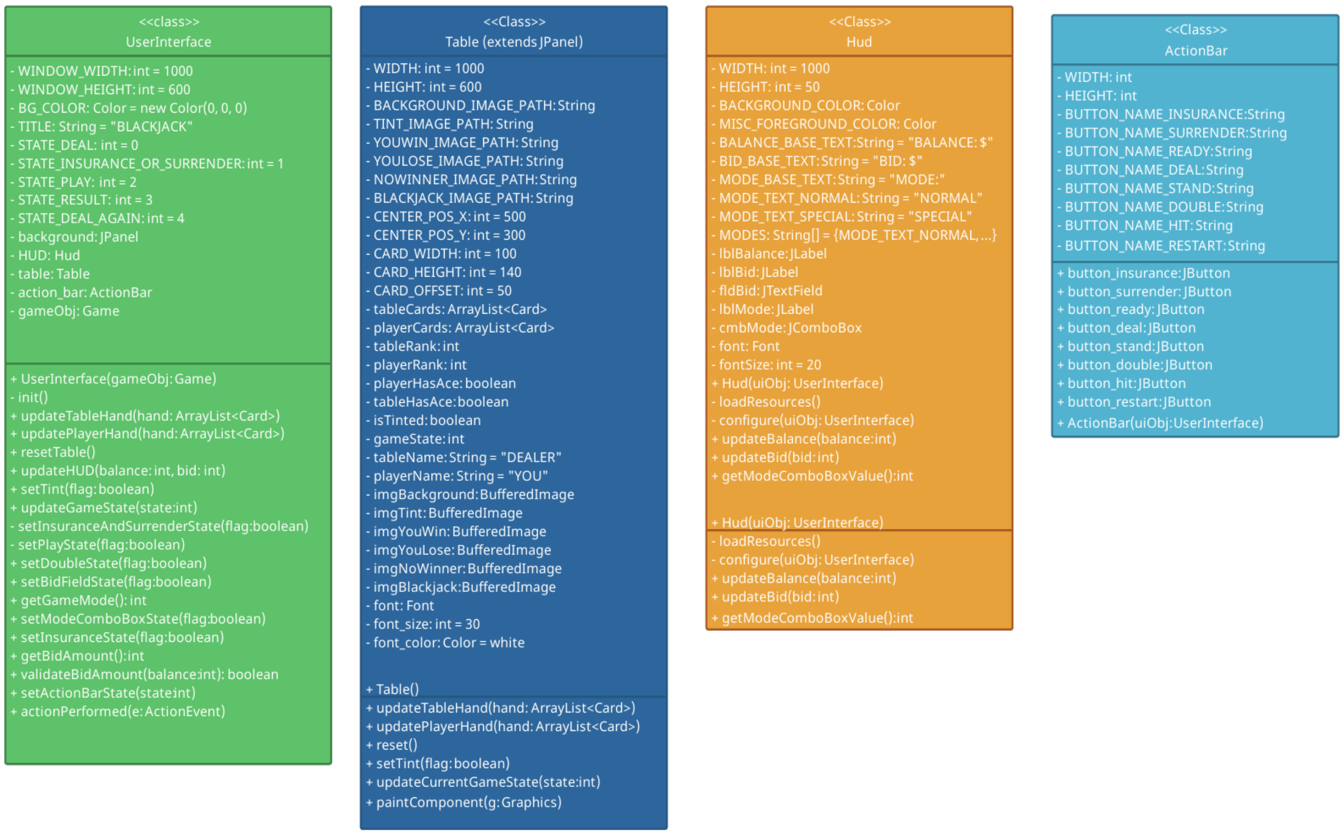
Class **Game** contains an instance of **UserInterface** class, two instances of **PlayerEntity** class, and two instances of **Deck** class.

UserInterface contains an instance of **Hud, Table,** and **ActionBar**.

UML Game Logic Design



UML GUI Class Design



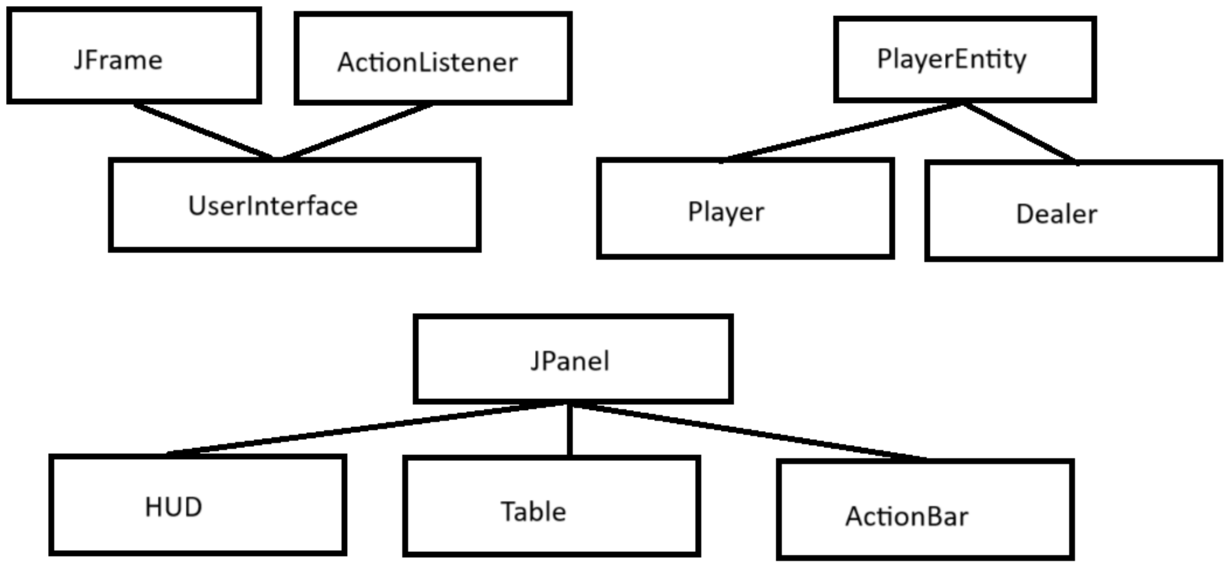
```
<<class>>
UserInterface
- WINDOW_WIDTH: int = 1000
- WINDOW_HEIGHT: int = 600
- BG_COLOR: Color = new Color(0, 0, 0)
- TITLE: String = "BLACKJACK"
- STATE_DEAL: int = 0
- STATE_INSURANCE_OR_SURRENDER: int = 1
- STATE_PLAY: int = 2
- STATE_RESULT: int = 3
- STATE_DEAL_AGAIN: int = 4
- background: JPanel
- HUD: Hud
- table: Table
- action_bar: ActionBar
- gameObj: Game
+ UserInterface(gameObj: Game)
- init()
+ updateTableHand(hand: ArrayList<Card>)
+ updatePlayerHand(hand: ArrayList<Card>)
+ resetTable()
+ updateHUD(balance: int, bid: int)
+ setTint(flag: boolean)
+ updateGameState(state:int)
- setInsuranceAndSurrenderState(flag:boolean)
- setPlayState(flag:boolean)
+ setDoubleState(flag:boolean)
+ setBidFieldState(flag:boolean)
+ getGameMode(): int
+ setModeComboBoxState(flag:boolean)
+ setInsuranceState(flag:boolean)
+ getBidAmount():int
+ validateBidAmount(balance:int): boolean
+ setActionBarState(state:int)
+ actionPerformed(e: ActionEvent)
```

```
<<Class>>
Table (extends JPanel)
- WIDTH: int = 1000
- HEIGHT: int = 600
- BACKGROUND_IMAGE_PATH: String
- TINT_IMAGE_PATH: String
- YOUWIN_IMAGE_PATH: String
- YOULOSE_IMAGE_PATH: String
- NOWINNER_IMAGE_PATH: String
- BLACKJACK_IMAGE_PATH: String
- CENTER_POS_X: int = 500
- CENTER_POS_Y: int = 300
- CARD_WIDTH: int = 100
- CARD_HEIGHT: int = 140
- CARD_OFFSET: int = 50
- tableCards: ArrayList<Card>
- playerCards: ArrayList<Card>
- tableRank: int
- playerRank: int
- playerHasAce: boolean
- tableHasAce: boolean
- isTinted: boolean
- gameState: int
- tableName: String = "DEALER"
- playerName: String = "YOU"
- imgBackground: BufferedImage
- imgTint: BufferedImage
- imgYouWin: BufferedImage
- imgYouLose: BufferedImage
- imgNoWinner: BufferedImage
- imgBlackjack:BufferedImage
- font: Font
- font_size: int = 30
- font_color: Color = white

+ Table()
+ updateTableHand(hand: ArrayList<Card>)
+ updatePlayerHand(hand: ArrayList<Card>)
+ reset()
+ setTint(flag: boolean)
+ updateCurrentGameState(state:int)
+ paintComponent(g: Graphics)
```

```
<<Class>>
Hud
- WIDTH: int = 1000
- HEIGHT: int = 50
- BACKGROUND_COLOR: Color
- MISC_FOREGROUND_COLOR: Color
- BALANCE_BASE_TEXT:String = "BALANCE: $"
- BID_BASE_TEXT: String = "BID: $"
- MODE_BASE_TEXT: String = "MODE:"
- MODE_TEXT_NORMAL: String = "NORMAL"
- MODE_TEXT_SPECIAL: String = "SPECIAL"
- MODES: String[] = {MODE_TEXT_NORMAL, ...}
- lblBalance: JLabel
- lblBid: JLabel
- fldBid: JTextField
- lblMode: JLabel
- cmbMode: JComboBox
- font: Font
- fontSize: int = 20
+ Hud(uiObj: UserInterface)
- loadResources()
- configure(uiObj: UserInterface)
+ updateBalance(balance:int)
+ updateBid(bid:int)
+ getModeComboBoxValue():int

+ Hud(uiObj: UserInterface)
- loadResources()
- configure(uiObj: UserInterface)
+ updateBalance(balance:int)
+ updateBid(bid:int)
+ getModeComboBoxValue():int
```

```
<<Class>>
ActionBar
- WIDTH: int
- HEIGHT: int
- BUTTON_NAME_INSURANCE:String
- BUTTON_NAME_SURRENDER:String
- BUTTON_NAME_READY:String
- BUTTON_NAME_DEAL:String
- BUTTON_NAME_STAND:String
- BUTTON_NAME_DOUBLE:String
- BUTTON_NAME_HIT:String
- BUTTON_NAME_RESTART:String
+ button_insurance:JButton
+ button_surrender:JButton
+ button_ready:JButton
+ button_deal:JButton
+ button_stand:JButton
+ button_double:JButton
+ button_hit:JButton
+ button_restart:JButton
+ ActionBar(uiObj:UserInterface)
```

Inheritance Hierarchy:



- **UserInterface-** Handles the GUI window as well as subcomponents such as the HUD, Table, and ActionBar. It can also act in response to certain game events.

- **PlayerEntity-** superclass for Player and Dealer that has common attributes and methods shared among players.

- **Dealer-** inherits from PlayerEntity and has certain unique traits.

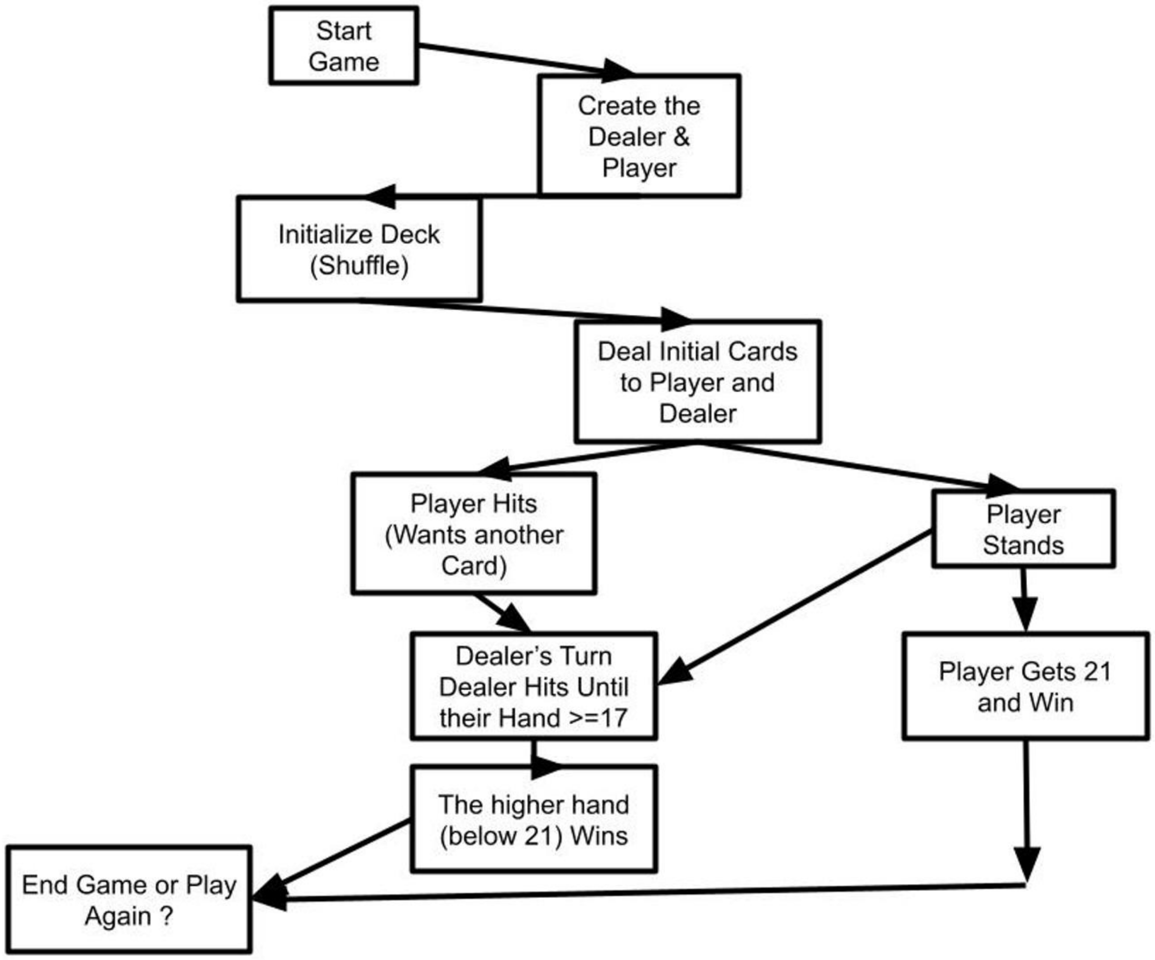
- **Player-** inherits from PlayerEntity and has certain unique traits.

- **HUD-** the Heads-Up-Display shown at the top of the screen (Mode, Balance, Bid) that inherits from JPanel.

- **Table-** shows the table with the cards and the scores; inherits from JPanel.

- **ActionBar-** the bar at the bottom of the screen that contains all the buttons; inherits from JPanel.

Use Case Flow Chart:


Start Game
Create the Dealer & Player
Initialize Deck (Shuffle)
Deal Initial Cards to Player and Dealer
Player Hits (Wants another Card)
Player Stands
Dealer's Turn Dealer Hits Until their Hand >=17
Player Gets 21 and Win
The higher hand (below 21) Wins
End Game or Play Again ?

Flow Chart Summary:

Summary of the Blackjack game Prototype

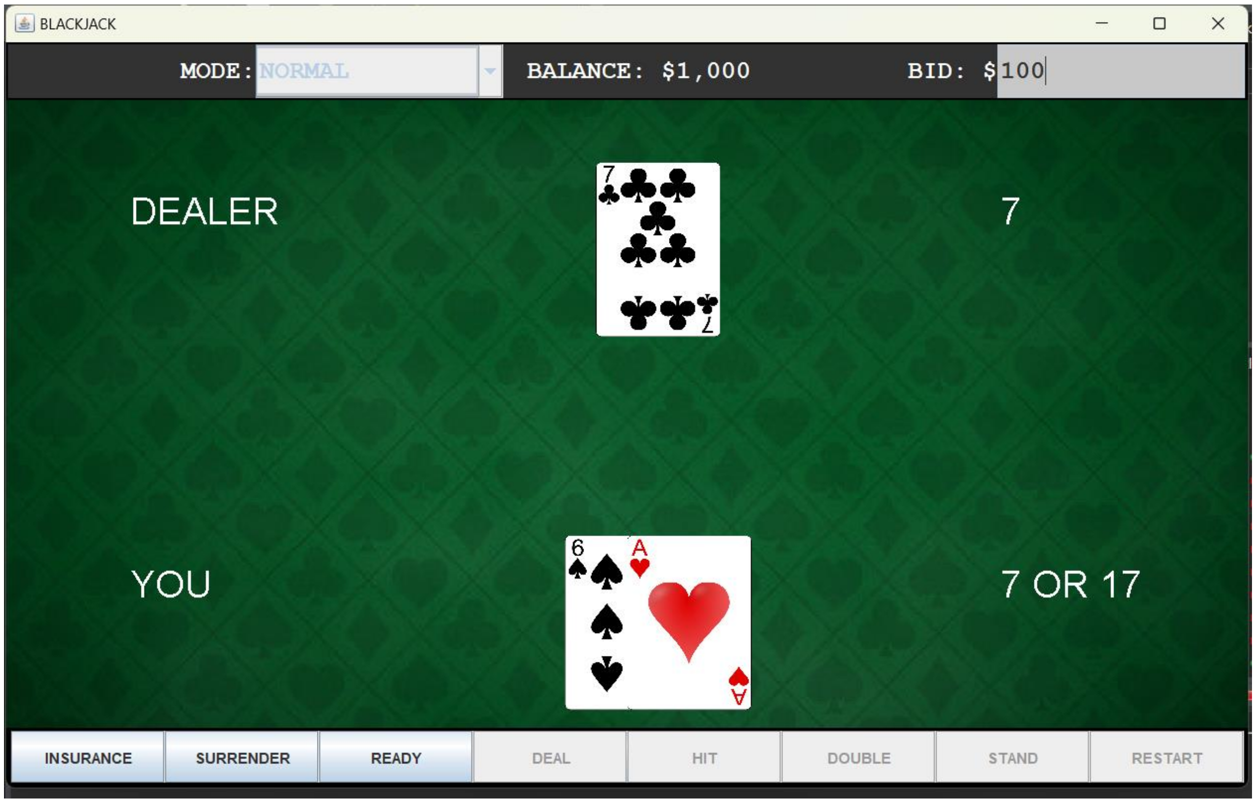
This Blackjack game prototype in Java includes the basic features necessary to play a simple version of Blackjack. The game flow involves:

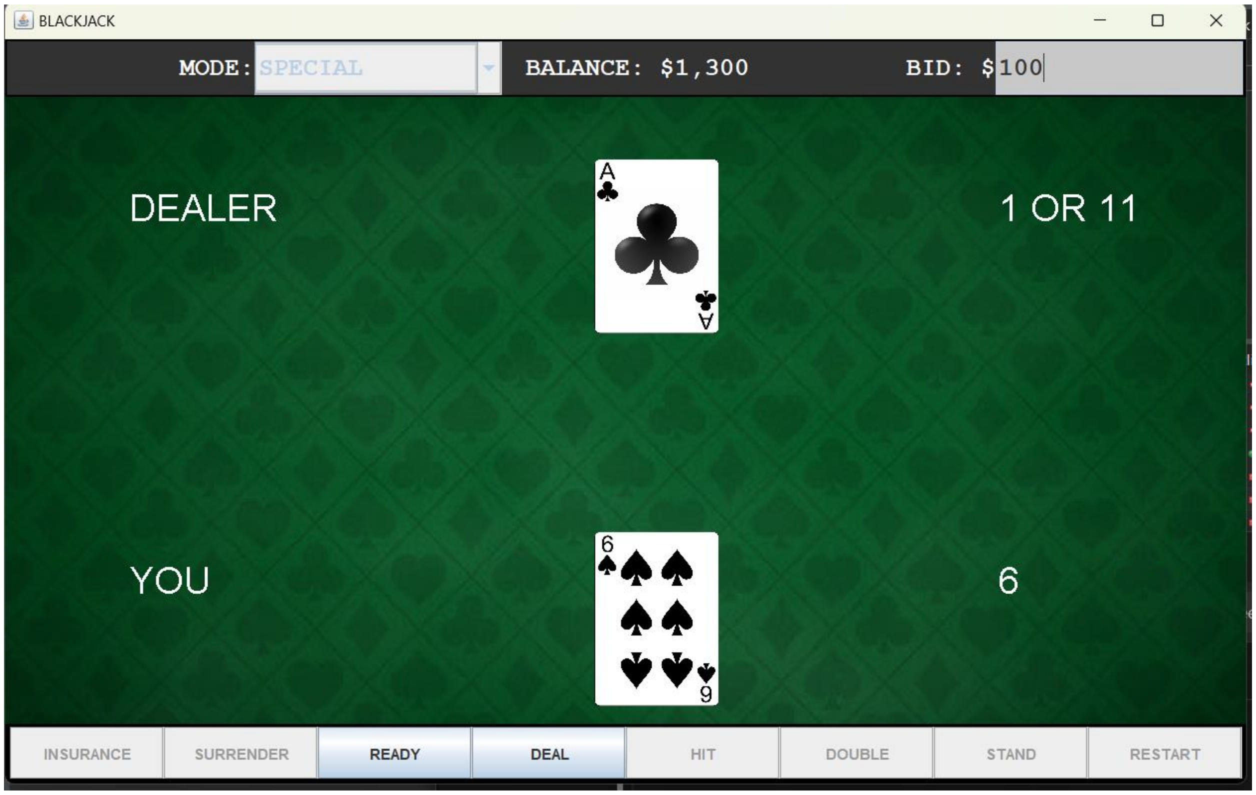
- Shuffling a deck of cards.
- Dealing two cards to the player and the dealer.
- Allowing the player to choose between hitting or standing.
- Implementing the dealer's behavior to automatically hit if their hand value is below 17.
- Determining the winner based on the final hand values.

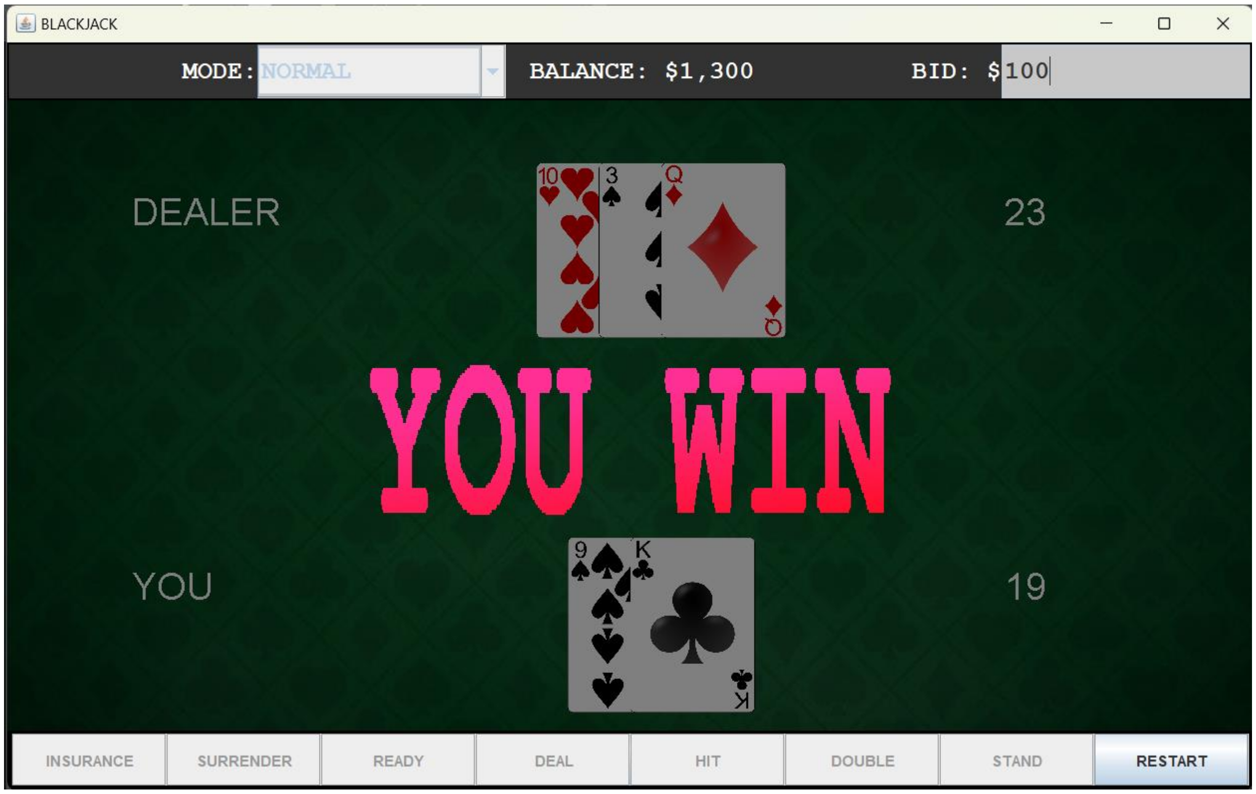
Key components of the game include the Card and Deck classes to represent and manipulate cards, and the game logic resides in the Game class, which handles the flow of the game. The program also incorporates simple mechanics such as Ace handling (allowing it to be worth 1 or 11).

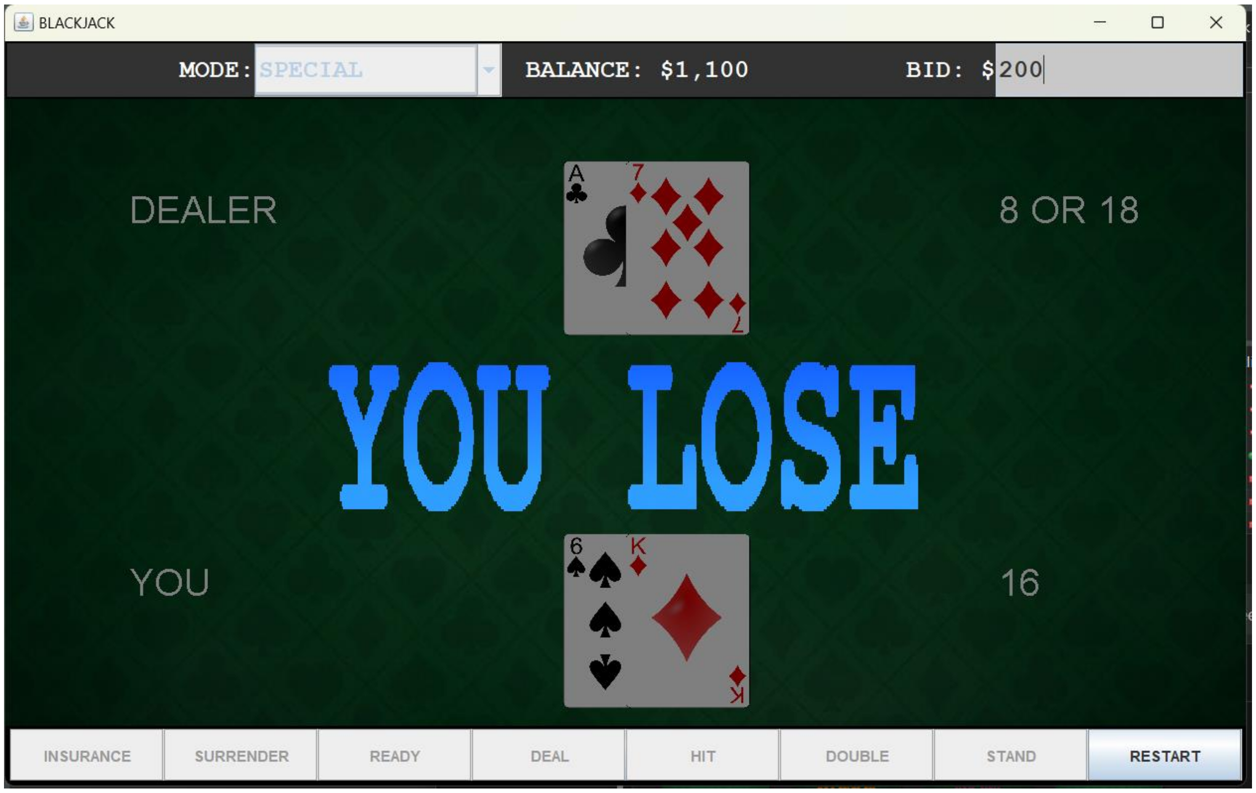
This prototype version of Blackjack can be expanded with features like multiple players, advanced betting, and more complex card behaviors.

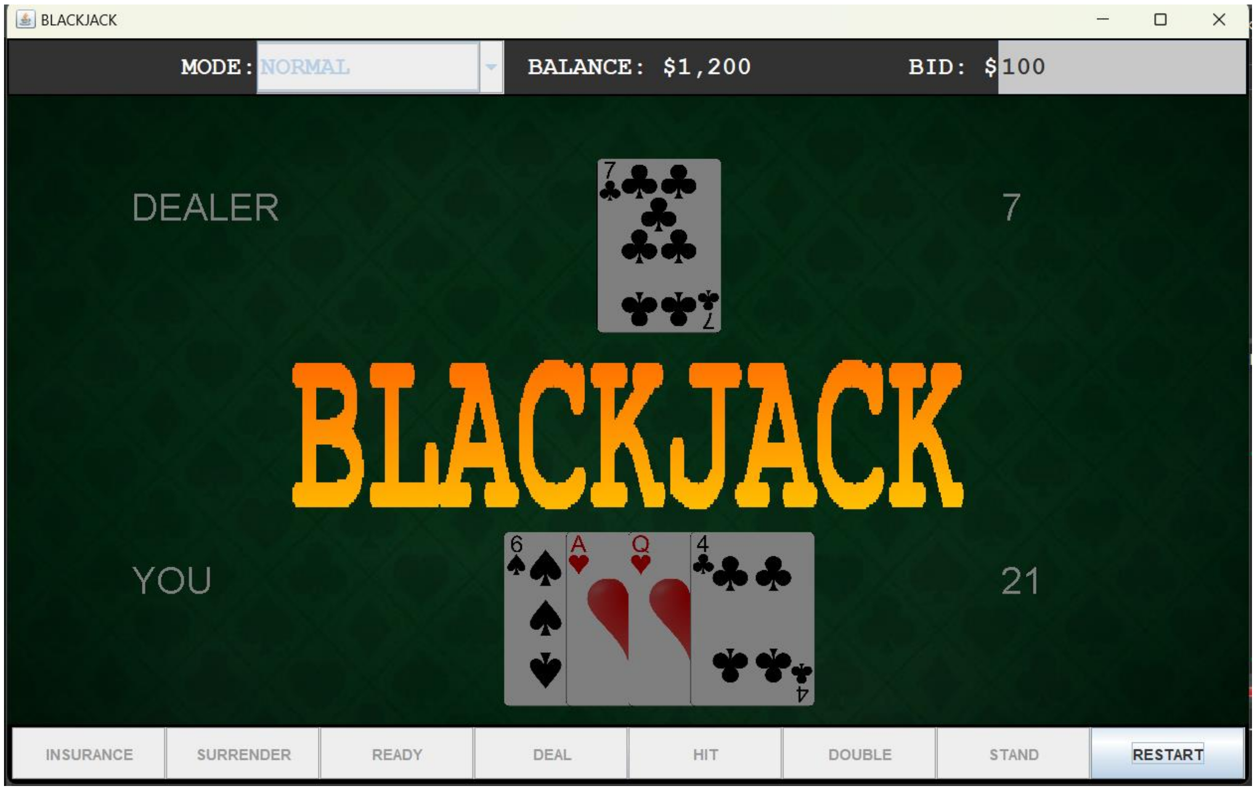
Screenshots:

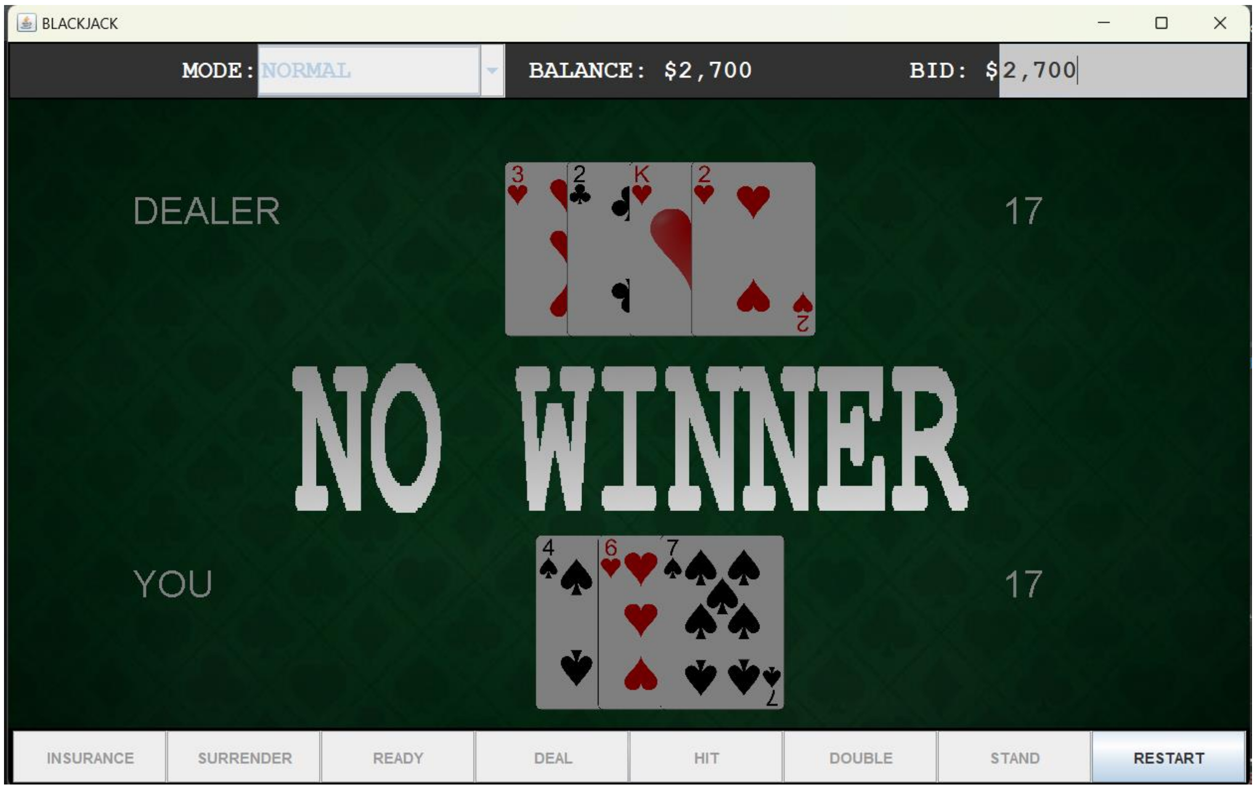



BLACKJACK
MODE: SPECIAL
BALANCE: $1,300
BID: $100
DEALER
1 OR 11
YOU
6
INSURANCE
SURRENDER
READY
DEAL
HIT
DOUBLE
STAND
RESTART


BLACKJACK
MODE: NORMAL
BALANCE: $1,300
BID: $100
DEALER
23
YOU WIN
YOU
19
INSURANCE
SURRENDER
READY
DEAL
HIT
DOUBLE
STAND
RESTART


BLACKJACK
MODE: SPECIAL
BALANCE: $1,100
BID: $200
DEALER
8 OR 18
YOU LOSE
YOU
16
INSURANCE
SURRENDER
READY
DEAL
HIT
DOUBLE
STAND
RESTART


BLACKJACK
MODE: NORMAL
BALANCE: $1,200
BID: $100
DEALER
7
BLACKJACK
YOU
21
INSURANCE
SURRENDER
READY
DEAL
HIT
DOUBLE
STAND
RESTART


BLACKJACK
MODE: NORMAL
BALANCE: $2,700
BID: $2,700
DEALER
17
NO WINNER
YOU
17
INSURANCE
SURRENDER
READY
DEAL
HIT
DOUBLE
STAND
RESTART

Conclusion

Future Enhancement:

This Blackjack game prototype in Java successfully simulates a simplified version of the card game. It demonstrates key concepts in object-oriented programming, such as using classes to represent cards, decks, and hands. The game implements basic mechanics such as card dealing, hand value calculation, and the player's decision-making process.

However, this prototype could be further developed in several ways:

- **Multiple Players**: Currently, the game supports only a single player. Adding multiple players and managing their turns would make the game more complex and engaging.
- **Advanced Betting System**: Implementing a new rule on betting system would introduce an additional layer of challenge and excitement. For example, other players can ask if they can bet on your card vs the dealer if they surrender their card. Adding more fun to the game.
- **Improving Graphical User Interface (GUI)**: Adding more visual details to the GUI like changing the background theme while playing would have a great, enjoyable user experience, allowing players to interact with the game more visually.
- **Advanced Dealer Rules**: More complex dealer behaviors, such as varying betting strategies or handling "double down" and "split" options, could be introduced.

Overall, the prototype lays a solid foundation for the Blackjack game and can be expanded with more advanced features to create a fully interactive and more feature-rich experience.

Appendix: Source Code

ActionBar.java

```
package blackjack;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JPanel;

public class ActionBar extends JPanel{

	private static final int WIDTH = 1000;
	private static final int HEIGHT = 50;

	private String BUTTON_NAME_INSURANCE = "INSURANCE";
	private String BUTTON_NAME_SURRENDER = "SURRENDER";
	private String BUTTON_NAME_READY = "READY";
	private String BUTTON_NAME_DEAL = "DEAL";
	private String BUTTON_NAME_STAND = "STAND";
	private String BUTTON_NAME_DOUBLE = "DOUBLE";
	private String BUTTON_NAME_HIT = "HIT";
	private String BUTTON_NAME_RESTART = "RESTART";

	//public to allow access to UserInterface class
	public JButton button_insurance;
	public JButton button_surrender;
	public JButton button_ready;
	public JButton button_deal;
	public JButton button_stand;
	public JButton button_double;
	public JButton button_hit;
	public JButton button_restart;

	public ActionBar(UserInterface uiObj)
	{
		setLayout(new GridLayout());
		setPreferredSize(new Dimension(WIDTH, HEIGHT));
		setBackground(Color.black);
		setBorder(BorderFactory.createRaisedBevelBorder());

		button_insurance = new JButton(BUTTON_NAME_INSURANCE);
		button_surrender = new JButton(BUTTON_NAME_SURRENDER);
		button_ready = new JButton(BUTTON_NAME_READY);
		button_deal = new JButton(BUTTON_NAME_DEAL);
		button_stand = new JButton(BUTTON_NAME_STAND);
		button_double = new JButton(BUTTON_NAME_DOUBLE);
		button_hit = new JButton(BUTTON_NAME_HIT);
		button_restart = new JButton(BUTTON_NAME_RESTART);

		button_insurance.addActionListener(uiObj);
		button_surrender.addActionListener(uiObj);
		button_ready.addActionListener(uiObj);
		button_deal.addActionListener(uiObj);
		button_hit.addActionListener(uiObj);
		button_double.addActionListener(uiObj);
		button_stand.addActionListener(uiObj);
		button_restart.addActionListener(uiObj);

		add(button_insurance);
		add(button_surrender);
		add(button_ready);
		add(button_deal);
		add(button_hit);
		add(button_double);
		add(button_stand);
		add(button_restart);
	}
}
```

Card.java

```
package blackjack;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.*;

public class Card {
   private final Rank rank;
   private final Suite suite;
   private ImageIcon icon;
   private BufferedImage image;
```

```
  public Card(final Rank rank, final Suite suite) {
    this.rank = rank;
    this.suite = suite;
    //this.icon = new ImageIcon(getClass().getResource("images/cards/" + rank.getImageFileName() + "_of_" +
suite.getImageFileName() + ".png"));
    try {
                                    this.image = ImageIO.read(new File("images/cards/" + rank.getImageFileName() + "_of_" +
suite.getImageFileName() + ".png"));
                        } catch (IOException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                        }
  }

  public Rank getRank() {
    return rank;
  }

  public Suite getSuite() {
    return suite;
  }

  public ImageIcon getIcon() {
    return icon;
  }

  public BufferedImage getImage()
  {
            return image;
  }

  @Override
  public String toString() {
    return rank.toString() + suite.toString();
  }
}
```

Dealer.java

```
package blackjack;

public class Dealer extends PlayerEntity{
            public Dealer(String name)
            {
                        super(name);
            }
}
```

Deck.java

```
package blackjack;

import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class Deck {
  private final List<Card> cards = new LinkedList<>();

  private final Deck backupDeck;

  public Deck() {
    this(true, null);
  }

  public Deck(final boolean emptyDeck, final Deck deck) {
    backupDeck = deck;

    if (emptyDeck) {
        return;
    }
    for (final Suite suite : Suite.values()) {
        for (final Rank rank : Rank.values()) {
            cards.add(new Card(rank, suite));
        }
    }
  }

  public void shuffle() {
    Collections.shuffle(cards);
  }

  public int size() {
    return cards.size();
  }

  public Card take() {
    if (cards.isEmpty()) {
        if (backupDeck != null) {
            moveCards(backupDeck);
        }
    }
    if (cards.isEmpty()) {
        return null;
    }
    return cards.remove(0);
  }

  public Card peek() {
    if (cards.isEmpty()) {
        return null;
    }
    return cards.get(cards.size() - 1);
  }

  public void addCards(final List<Card> cards) {
    this.cards.addAll(cards);
  }

  //transfer all the returned cards back to the serving deck
  public void moveCards(final Deck deck) {
    this.cards.addAll(deck.cards);
    deck.cards.clear();
    shuffle();
  }

  @Override
  public String toString() {
    if (cards.isEmpty()) {
        return "Empty Deck";
    }

    boolean first = true;
    final StringBuilder sb = new StringBuilder();
    for (final Card card : cards) {
        if (first) {
            first = false;
        } else {
            sb.append(",");
```

```
      }
      sb.append(card);
    }

    return sb.toString();
  }
}
```

Game.java

```
package blackjack;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

public class Game{
	//=====================================================================
	//Objects
	//=====================================================================
	private UserInterface UI;

	//=====================================================================
	//Constants
	//=====================================================================
	private static final int INITIAL_BALANCE = 1000;
	private static final int INITIAL_BID = 100;

	//maximum number of cards possible
	// 4 aces, 4 2's, and 3 3's -> 11 cards, rank = 4 + 8 + 9 = 21
	private static final int MAX_CARDS = 11;

	//allow access to Table class
	public static final int GAME_STATE_ONGOING = 0;
	public static final int GAME_STATE_YOUWIN = 1;
	public static final int GAME_STATE_YOULOSE = 2;
	public static final int GAME_STATE_NOWINNER = 3;
	public static final int GAME_STATE_BLACKJACK = 4;

	//game modes
	public static final int GAME_MODE_NORMAL = 0;
	//(J, Q, K) are (11, 12, 13) respectively
	//and you have the choice to only draw one card
	public static final int GAME_MODE_SPECIAL = 1;

	//=====================================================================
	//Game-Specific variables
	//=====================================================================
	private Deck returnCardDeck = new Deck();
	private Deck servingCardDeck = new Deck(false, returnCardDeck);

	final ArrayList<Card> tableCards = new ArrayList<>(); // cards table has
	final ArrayList<Card> playerCards = new ArrayList<>(); // cards player has

	private int balance;	//if this goes to 0 then its game over
	private int bid; // current bid
	private boolean purchasedInsurance; // did player earned insurance?

	private Player player = new Player("YOU");
	private Dealer table = new Dealer("DEALER");

	private int gameMode;

	public Game()
	{
		init();
	}

	private void init()
	{
		//start UI
		UI = new UserInterface(this);

		balance = INITIAL_BALANCE;

		UI.setNames(player.name, table.name);

		//start new round
		reset();
	}

	//start new round
	private void reset()
	{
		// game has not started -> no insurance, no ace dealt, ranks are 0
		purchasedInsurance = false;

		player.setHasAce(false);
		table.setHasAce(false);
		player.setHasDrawn(false);

		bid = INITIAL_BID;	//set back to default every round

		returnCardDeck.addCards(player.cards);
		returnCardDeck.addCards(table.cards);
		player.cards.clear();
		table.cards.clear();
		servingCardDeck.moveCards(returnCardDeck);		//move returned cards back to serving deck and shuffle

		UI.setModeComboBoxState(true);		//enable drop down
		UI.setBidFieldState(true);		//enable bid field
		UI.setActionBarState(UserInterface.STATE_DEAL);
		UI.resetTable();
		UI.updateHUD(balance, bid);
		UI.setTint(false);
		UI.updateGameState(GAME_STATE_ONGOING);
	}

	//change the current game mode and alter the values of face cards
	private void changeGameMode(int gameMode)
	{
		try
		{
			this.gameMode = gameMode;
			if(this.gameMode == GAME_MODE_NORMAL)
			{
				Rank.Jack.changeNumericValue(10);
				Rank.Queen.changeNumericValue(10);
				Rank.King.changeNumericValue(10);
			}
			else if(this.gameMode == GAME_MODE_SPECIAL)
			{
				Rank.Jack.changeNumericValue(11);
				Rank.Queen.changeNumericValue(12);
				Rank.King.changeNumericValue(13);
			}
			else
			{
				throw new IllegalArgumentException();
			}
```

```
        }catch(Exception e)
        {
            System.out.println("Invalid game mode");
        }
    }

    //make the player draw a card
    private void playerDrawCard()
    {
        final Card playerCard = servingCardDeck.take();     //player is dealt a card from the serving deck
        player.cards.add(playerCard);
        if (playerCard.getRank().equals(Rank.Ace))          //if the card is an ace set flag
        {
            player.setHasAce(true);
        }
    }

    // logic to handle table's turn
    private void tableTurn() {
        // if table's rank is less than or 16, it continues to draw a card
        // otherwise stays
        // for this assume ace is 11
        // finally, a winner/loser is decided and game ends
        int tableRankFinal = table.getRank() + (table.getHasAce() ? 10 : 0);

        for (int i = 2  ; i < MAX_CARDS && tableRankFinal <= 16; i++)      //keep drawing until max cards or over 16
        {
            final Card tableCard = servingCardDeck.take();   //draw a card
            table.cards.add(tableCard);

            if (!table.getHasAce() && tableCard.getRank().equals(Rank.Ace)) {
                table.setHasAce(true);
                tableRankFinal += 10;
            }
            tableRankFinal += tableCard.getRank().getNumericValue();
        }

        UI.updateTableHand(table.cards);          //update the screen
    }

    private void blackjack()
    {
        balance += 2 * bid;

        UI.setActionBarState(UserInterface.STATE_RESULT);
        UI.setTint(true);
        UI.updateHUD(balance, bid);
        UI.updateGameState(GAME_STATE_BLACKJACK);
    }

    //determines winner or loser
    private void determineWinner() {
        final int playerScore = player.computeScore();
        final int tableScore = table.computeScore();

        if (playerScore > 21) {
            // if player score is > 21, doesn't matter what is the
            // score of the table, player loses

            balance -= bid;

            UI.updateGameState(GAME_STATE_YOULOSE);
        } else if (tableScore > 21) {
            // else, if table went overboard, player is the winner

            balance += bid;
```

```
                UI.updateGameState(GAME_STATE_YOUWIN);
            } else if (playerScore == tableScore) {
                // both are less than 21, but same score, no winner
                UI.updateGameState(GAME_STATE_NOWINNER);
            } else if (playerScore > tableScore) {
                // player has the higher score, player wins winner

                balance += bid;

                UI.updateGameState(GAME_STATE_YOUWIN);
            } else if(tableScore == 21){
                // table scored a blackjack so you lose
                if(purchasedInsurance)          //if insurance was bought, then only lose half of the bid
                {
                    bid /= 2;
                }
                balance -= bid;

                UI.updateGameState(GAME_STATE_YOULOSE);
            } else { // if (tableScore > playerScore) ...
                // table has higher score, so player is the loser

                balance -= bid;

                UI.updateGameState(GAME_STATE_YOULOSE);
            }

            //update the value of balance

            UI.setActionBarState(UserInterface.STATE_RESULT);
            UI.updateHUD(balance, bid);
            UI.setTint(true);
        }

        //=======================================================================
        //Button callback methods
        //=======================================================================

        //when the deal button is clicked
        public void onDeal()
        {
            changeGameMode(UI.getGameMode());

            //only do this the first time
            if(!player.getHasDrawn())
            {
                if(!UI.validateBidAmount(balance)) return;
                bid = UI.getBidAmount();

                UI.setModeComboBoxState(false);         //disable drop down
                UI.setBidFieldState(false);       //disable bid field
                UI.setActionBarState(UserInterface.STATE_INSURANCE_OR_SURRENDER);

                UI.resetTable();

                //table draws one card
                final Card tableCard = servingCardDeck.take();
                if(tableCard.getRank().equals(Rank.Ace))
                {
                    table.setHasAce(true);
                }
                table.cards.add(tableCard);
                UI.updateTableHand(table.cards);
            }
            if(gameMode == GAME_MODE_SPECIAL)  //only draw one card at a time if game mode is SPECIAL
            {
                playerDrawCard();
                //UI.updatePlayerHand(playerCards);
                UI.updatePlayerHand(player.cards);

                //if(!playerHasDrawn)             //if this is the first time
                if(!player.getHasDrawn())
                {
                    UI.setActionBarState(UserInterface.STATE_DEAL_AGAIN);       //click deal one more time to
    draw another card
                    Util.delay(250);
                }
                else        //second time
                {
                    bid *= 2;    //double bid
                    UI.updateHUD(balance, bid);
                    UI.setActionBarState(UserInterface.STATE_INSURANCE_OR_SURRENDER);
                    if(purchasedInsurance)          //disable the insurance button after once
                    {
                        UI.setInsuranceState(false);
                    }
                }

                //playerHasDrawn = true;
                player.setHasDrawn(true);

            }
            else        //NORMAL MODE
            {
                //player draws two cards
                for(int i = 0;i < 2;i++)
                {
                    playerDrawCard();
                }
                //UI.updatePlayerHand(playerCards);
                UI.updatePlayerHand(player.cards);
            }

            //check score for blackjack
            if(player.getRank() == 21 || (player.getRank() == 11 && player.getHasAce()))
            {
                blackjack();
            }
        }

        //when the hit button is clicked
        public void onHit()
        {
            UI.setActionBarState(UserInterface.STATE_PLAY);

            playerDrawCard();

            //draw the new rank and disable the double button
            UI.updatePlayerHand(player.cards);
            UI.setDoubleState(false);

            // if player goes over 21, then it's a bust
            if(player.getRank() > 21)
            {
                UI.setActionBarState(UserInterface.STATE_RESULT);

                determineWinner();
            }
            else if(player.getRank() == 21 || (player.getRank() == 11 && player.getHasAce()))
            {
```

```
                    blackjack();
                }
        }

        //when the double button is clicked
        public void onDouble()
        {
                bid *= 2;
                UI.updateHUD(balance, bid);

                playerDrawCard();
                //UI.updatePlayerHand(playerCards);
                UI.updatePlayerHand(player.cards);

                tableTurn();

                determineWinner();
        }

        //when the stand button is clicked
        public void onStand()
        {
                tableTurn();
                determineWinner();
        }

        //when the insurance button is clicked
        public void onInsurance()
        {
                bid *= 2;
                UI.updateHUD(balance, bid);

                purchasedInsurance = true;

                if(purchasedInsurance)          //disable the insurance button after once
                {
                        UI.setInsuranceState(false);
                }
        }

        //when the surrender button is clicked
        public void onSurrender()
        {
                balance -= bid;

                UI.updateGameState(GAME_STATE_YOULOSE);

                UI.setActionBarState(UserInterface.STATE_RESULT);
                UI.updateHUD(balance, bid);
                UI.setTint(true);
        }

        //when the ready button is clicked
        public void onReady()
        {
                UI.setActionBarState(UserInterface.STATE_PLAY);
        }

        //when the restart button is clicked
        public void onRestart()
        {
                reset();
        }
}
```

Hud.java

```java
package blackjack;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Hud extends JPanel{

		private static final int WIDTH = 1000;
		private static final int HEIGHT = 50;
		private static final Color BACKGROUND_COLOR = new Color(50, 50, 50);  //gray
		private static final Color MISC_FOREGROUND_COLOR = new Color(200, 200, 200);   //light gray

		private static final String BALANCE_BASE_TEXT = "BALANCE: $";
		private static final String BID_BASE_TEXT = "BID: $";
		private static final String MODE_BASE_TEXT = "MODE:";
		private static final String MODE_TEXT_NORMAL = "NORMAL";
		private static final String MODE_TEXT_SPECIAL = "SPECIAL";
		private static final String[] MODES = {MODE_TEXT_NORMAL, MODE_TEXT_SPECIAL};//for the drop down

		private JLabel lblBalance;
		private JLabel lblBid;
		public JTextField fldBid;		//public to allow access to UserInterface
		private JLabel lblMode;
		public JComboBox cmbMode;

		private Font font;
		private static final int fontSize = 20;

		public Hud(UserInterface uiObj)
		{
				loadResources();
				configure(uiObj);
		}

		//load font
		private void loadResources()
		{
				font = new Font("Courier New Bold", Font.PLAIN, fontSize);
		}

		//configure the components
		private void configure(UserInterface uiObj)
		{
				setPreferredSize(new Dimension(WIDTH, HEIGHT));
				setBackground(Color.black);
				setBorder(BorderFactory.createRaisedBevelBorder());
				setLayout(new GridLayout());

				lblMode = new JLabel(MODE_BASE_TEXT);
				lblBalance = new JLabel(BALANCE_BASE_TEXT);
				lblBid = new JLabel(BID_BASE_TEXT);
				fldBid = new JTextField();
				cmbMode = new JComboBox(MODES);

				lblMode.setOpaque(true);
				lblMode.setBackground(BACKGROUND_COLOR);
		lblMode.setForeground(Color.white);
		lblMode.setFont(font);
		lblMode.setHorizontalAlignment(JLabel.RIGHT);
		cmbMode.setOpaque(true);
		cmbMode.setBackground(MISC_FOREGROUND_COLOR);
		cmbMode.setFont(font);
		lblBalance.setOpaque(true);
		lblBalance.setBackground(BACKGROUND_COLOR);
		lblBalance.setForeground(Color.white);
		lblBalance.setFont(font);
		lblBalance.setHorizontalAlignment(JLabel.RIGHT);
		lblBid.setOpaque(true);
		lblBid.setBackground(BACKGROUND_COLOR);
		lblBid.setForeground(Color.white);
		lblBid.setFont(font);
		lblBid.setText(BID_BASE_TEXT);
		lblBid.setHorizontalAlignment(JLabel.RIGHT);
		fldBid.setOpaque(true);
		fldBid.setBackground(MISC_FOREGROUND_COLOR);
		fldBid.setFont(font);

		fldBid.addActionListener(uiObj);

		add(lblMode);
		add(cmbMode);
		add(lblBalance);
		add(lblBid);
		add(fldBid);
	}

	//update the balance value shown on the screen
	public void updateBalance(int balance)
	{
		lblBalance.setText(BALANCE_BASE_TEXT + String.format("%,d", balance));
	}

	//update the bid amount shown in the text field
	public void updateBid(int bid)
	{
		fldBid.setText(String.format("%,d", bid));
	}

	//return the selected index of the drop down menu
	public int getModeComboBoxValue()
	{
		return cmbMode.getSelectedIndex();
	}
}
```

Player.java

```
package blackjack;

public class Player extends PlayerEntity{
	private boolean hasDrawn;

	public Player(String name)
	{
		super(name);
	}

	public boolean getHasDrawn()
	{
		return hasDrawn;
	}
	public void setHasDrawn(boolean flag)
	{
		hasDrawn = flag;
	}
}
```

PlayerEntity.java

```
package blackjack;

import java.util.ArrayList;

public class PlayerEntity {
	public ArrayList<Card> cards = new ArrayList<>();
	protected boolean hasAce;
	protected String name;

	public PlayerEntity(String name)
	{
		this.name = name;
	}

	public boolean getHasAce()
	{
		return hasAce;
	}

	public void setHasAce(boolean flag)
	{
		hasAce = flag;
	}

	//will always treat Ace as 1
	public int getRank()
	{
		return Util.calculateHandRank(cards);
	}

	public int computeScore()
	{
		int rank = getRank();
		if (hasAce) {
			return (rank + 10) > 21 ? rank : rank + 10;
		} else {
			return rank;
		}
	}
}
```

Rank.java

```
package blackjack;

public enum Rank {
	Ace(1, "A", "ace"),
	Two(2, "2", "2"),
	Three(3, "3", "3"),
	Four(4, "4", "4"),
	Five(5, "5", "5"),
	Six(6, "6", "6"),
	Seven(7, "7", "7"),
	Eight(8, "8", "8"),
	Nine(9, "9", "9"),
	Ten(10, "10", "10"),
		Jack(10, "J", "jack"),
		Queen(10, "Q", "queen"),
		King(10, "K", "king");

		private int numericValue;		//will change depending on the game mode
		private final String printValue;
		private final String imageFileNameComponent;

		private Rank(final int numericValue, final String printValue, final String imageFileNameComponent)
		{
			this.numericValue = numericValue;
			this.printValue = printValue;
			this.imageFileNameComponent = imageFileNameComponent;
		}

		//use this to change the values of face cards for different game modes
		public void changeNumericValue(int newVal)
		{
			this.numericValue = newVal;
		}

		public int getNumericValue() {
			return numericValue;
		}

		public String getImageFileName() {
			return imageFileNameComponent;
		}

		@Override
		public String toString() {
			return printValue;
		}
}
```

Suite.java

```
package blackjack;

public enum Suite {
  // https://en.wikipedia.org/wiki/Playing_card#Symbols_in_Unicode
  HEART("\u2661", "hearts"),
  DIAMOND("\u2662", "diamonds"),
  CLUB("\u2667", "clubs"),
  SPADE("\u2664", "spades");

  private final String printValue;
  private final String imageFileNameComponent;

  Suite(final String printValue, final String imageFileNameComponent) {
    this.printValue = printValue;
    this.imageFileNameComponent = imageFileNameComponent;
  }

  public String getImageFileName() {
    return imageFileNameComponent;
  }

  @Override
  public String toString() {
    return printValue;
  }
}
```

Table.java

```
package blackjack;

import javax.imageio.ImageIO;
import java.awt.font.*;
import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

public class Table extends JPanel{

	//======================================================================
	//Constants
	//======================================================================
	private static final int WIDTH = 1000;
	private static final int HEIGHT = 600;
	private static final String BACKGROUND_IMAGE_PATH = "images/background.jpg";
	private static final String TINT_IMAGE_PATH = "images/tint.png";
	private static final String YOUWIN_IMAGE_PATH = "images/banners/youwin.png";
	private static final String YOULOSE_IMAGE_PATH = "images/banners/youlose.png";
	private static final String NOWINNER_IMAGE_PATH = "images/banners/nowinner.png";
	private static final String BLACKJACK_IMAGE_PATH = "images/banners/blackjack.png";

	private static final int CENTER_POS_X = WIDTH / 2;
	private static final int CENTER_POS_Y = HEIGHT / 2;

	private static final int TABLE_CARDS_POS_X = 200;
	private static final int TABLE_CARDS_POS_Y = 50;
	private static final int PLAYER_CARDS_POS_X = 200;
	private static final int PLAYER_CARDS_POS_Y = 350;

	private static final int CARD_WIDTH = 100;
	private static final int CARD_HEIGHT = 140;
	private static final int CARD_OFFSET = CARD_WIDTH / 2;

	private static final int TABLE_RANK_POS_X = 800;
	private static final int TABLE_RANK_POS_Y = 100;
	private static final int PLAYER_RANK_POS_X = 800;
	private static final int PLAYER_RANK_POS_Y = 400;

	private static final int TABLE_NAME_POS_X = 100;
	private static final int TABLE_NAME_POS_Y = 100;
	private static final int PLAYER_NAME_POS_X = 100;
	private static final int PLAYER_NAME_POS_Y = 400;

	//======================================================================
	//Resources
	//======================================================================
	private BufferedImage imgBackground;
	private BufferedImage imgTint;
	private BufferedImage imgYouWin;
	private BufferedImage imgYouLose;
	private BufferedImage imgNoWinner;
	private BufferedImage imgBlackjack;
	private Font font;
	private int font_size = 30;
	private Color font_color = Color.white;

	private ArrayList<Card> tableCards = new ArrayList<>();
	private ArrayList<Card> playerCards = new ArrayList<>();

	private int tableRank;
	private int playerRank;

	private boolean playerHasAce = false;
	private boolean tableHasAce = false;

	private boolean isTinted = false;

	private int gameState;		//set by UserInterface, set by Game

	private String tableName = "";
	private String playerName = "";

	public Table()
	{
		loadResources();
		configure();
	}

	//load images and fonts
	private void loadResources()
	{
		imgBackground = Util.loadImage(BACKGROUND_IMAGE_PATH);
		imgTint = Util.loadImage(TINT_IMAGE_PATH);
		imgYouWin = Util.loadImage(YOUWIN_IMAGE_PATH);
		imgYouLose = Util.loadImage(YOULOSE_IMAGE_PATH);
		imgNoWinner = Util.loadImage(NOWINNER_IMAGE_PATH);
		imgBlackjack = Util.loadImage(BLACKJACK_IMAGE_PATH);

		font = new Font("TimesRoman", Font.PLAIN, font_size);
	}

	//configure the JPanel
	private void configure()
	{
		setPreferredSize(new Dimension(WIDTH, HEIGHT));
	}

	public void setNames(String playerName, String dealerName)
	{
		this.playerName = playerName;
		this.tableName = dealerName;
	}

	//update the cards shown on screen for the table
	public void updateTableHand(ArrayList<Card> hand)
	{
		tableCards = hand;
		tableRank = Util.calculateHandRank(tableCards);	//update the displayed table rank
		tableHasAce = Util.hasAce(tableCards);
		repaint();
	}

	//update the cards shown on screen for the player
	public void updatePlayerHand(ArrayList<Card> hand)
	{
		playerCards = hand;
		playerRank = Util.calculateHandRank(playerCards);
		playerHasAce = Util.hasAce(playerCards);
		repaint();
	}

	//reset cards and ranks
	public void reset()
	{
		tableCards.clear();
		playerCards.clear();
```

```
			playerRank = tableRank =  0;

			playerHasAce = tableHasAce = false;

			repaint();
		}

		//tint the screen
		public void setTint(boolean flag)
		{
			isTinted = flag;
			repaint();
		}

		//used to determine which banner to show on the result screen
		public void updateCurrentGameState(int state)
		{
			gameState = state;
			repaint();
		}

		@Override
		public void paintComponent(Graphics g)  //draw the scene and the cards
		{
			super.paintComponent(g);
			g.drawImage(imgBackground, 0, 0, WIDTH, HEIGHT, 0, 0, imgBackground.getWidth(), imgBackground.getHeight(),
null);

			//draw table's hand
			int centered_x = CENTER_POS_X - ((CARD_OFFSET * tableCards.size()) / 2);		//center the cards
			for(int i = 0;i < tableCards.size();i++)
			{
				Card card = tableCards.get(i);
				int offset = (i * CARD_OFFSET);  //slight overlap on each card
				g.drawImage(card.getImage(),
						centered_x + offset, TABLE_CARDS_POS_Y,		//top left
						centered_x + offset + CARD_WIDTH, TABLE_CARDS_POS_Y + CARD_HEIGHT,
		//bottom right
						0, 0, card.getImage().getWidth(), card.getImage().getHeight(), //source clip to draw
from
						null);
			}
			//draw player's hand
			centered_x = CENTER_POS_X - ((CARD_OFFSET * playerCards.size()) / 2); //center the cards
			for(int i = 0;i < playerCards.size();i++)
			{
				Card card = playerCards.get(i);
				int offset = (i * CARD_OFFSET);  //slight overlap on each card
				g.drawImage(card.getImage(),
						centered_x + offset, PLAYER_CARDS_POS_Y,
						centered_x + offset + CARD_WIDTH, PLAYER_CARDS_POS_Y + CARD_HEIGHT,
						0, 0, card.getImage().getWidth(), card.getImage().getHeight(),
						null);
			}

			g.setFont(font);
			g.setColor(font_color);

			//draw player names
			g.drawString(tableName, TABLE_NAME_POS_X, TABLE_NAME_POS_Y);
			g.drawString(playerName, PLAYER_NAME_POS_X, PLAYER_NAME_POS_Y);

			//draw table and player ranks
			String tableRankStr = String.valueOf(tableRank);
			if(tableHasAce && (tableRank + 10 <= 21))
			{
					tableRankStr = String.valueOf(tableRank) + " OR " + String.valueOf(tableRank + 10);
				}
				String playerRankStr = String.valueOf(playerRank);
				if(playerHasAce && (playerRank + 10 <= 21))
				{
					playerRankStr = String.valueOf(playerRank) + " OR " + String.valueOf(playerRank + 10);
				}
				g.drawString(tableRankStr, TABLE_RANK_POS_X, TABLE_RANK_POS_Y);
				g.drawString(playerRankStr, PLAYER_RANK_POS_X, PLAYER_RANK_POS_Y);

				if(isTinted)
				{
					g.drawImage(imgTint,
							0, 0, WIDTH, HEIGHT,
							0, 0, imgTint.getWidth(), imgTint.getHeight(),
							null);
				}

				//show result banner
				BufferedImage banner = null;
				switch(gameState)
				{
				case Game.GAME_STATE_ONGOING:
					break;
				case Game.GAME_STATE_YOUWIN:
					banner = imgYouWin;
					break;
				case Game.GAME_STATE_YOULOSE:
					banner = imgYouLose;
					break;
				case Game.GAME_STATE_NOWINNER:
					banner = imgNoWinner;
					break;
				case Game.GAME_STATE_BLACKJACK:
					banner = imgBlackjack;
					break;
				default:
					;
				}
				if(banner != null)	//don't show banner if game is still ongoing
				{
					g.drawImage(banner,
							CENTER_POS_X - (banner.getWidth() / 2), CENTER_POS_Y - banner.getHeight() - 50,
							CENTER_POS_X - (banner.getWidth() / 2) + banner.getWidth(), CENTER_POS_Y -
(banner.getHeight() / 2) + banner.getHeight(),
							0, 0, banner.getWidth(), banner.getHeight(),
							null);
				}
		}
}
```

UserInterface.java

```
package blackjack;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.*;

public class UserInterface extends JFrame implements ActionListener{
	//==========================================================================
	//Constants
	//==========================================================================

	//These two might not be necessary
	public static final int WINDOW_WIDTH = 1000;
	public static final int WINDOW_HEIGHT = 600;

	private Color BG_COLOR = new Color(0, 0, 0);

	private String TITLE = "BLACKJACK";

	//action bar states
	public static final int STATE_DEAL = 0;
	public static final int STATE_INSURANCE_OR_SURRENDER = 1;
	public static final int STATE_PLAY = 2;
	public static final int STATE_RESULT = 3;
	public static final int STATE_DEAL_AGAIN = 4;

	//==========================================================================
	//Objects
	//==========================================================================
	private JPanel background;		//common background
	//Custom objects
	private Hud HUD;
	private Table table;	//basically a canvas (extends JPanel)
	private ActionBar action_bar;

	private Game gameObj;

	public UserInterface(Game gameObj)
	{
		this.gameObj = gameObj;

		init();
	}

	private void init()
	{
		setTitle(TITLE);

		//create components
		background = new JPanel();

		table = new Table();
		HUD = new Hud(this);
		action_bar = new ActionBar(this);

		//Configure components

		background.setBackground(BG_COLOR);
		background.setLayout(new BoxLayout(background, BoxLayout.PAGE_AXIS));
		background.setBounds(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);

		//Add components to containers

		background.add(HUD);
		background.add(table);
		background.add(action_bar);

		add(background);

		//configure frame

		setLayout(new BorderLayout());
```

```
			setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			getContentPane().setPreferredSize(new Dimension(WINDOW_WIDTH, WINDOW_HEIGHT));
			pack();
			setVisible(true);
		}

		public void setNames(String playerName, String dealerName)
		{
			table.setNames(playerName, dealerName);
		}

		//update the cards shown on the screen for the Table
		public void updateTableHand(ArrayList<Card> hand)
		{
			table.updateTableHand(hand);
		}

		//update the cards shown on the screen for the player
		public void updatePlayerHand(ArrayList<Card> hand)
		{
			table.updatePlayerHand(hand);
		}

		//reset the table back to initial states
		public void resetTable()
		{
			table.reset();
		}

		//show updated balance and bid
		public void updateHUD(int balance, int bid)
		{
			HUD.updateBalance(balance);
			HUD.updateBid(bid);
		}

		public void setTint(boolean flag)
		{
			table.setTint(flag);
		}

		public void updateGameState(int state)
		{
			table.updateCurrentGameState(state);
		}

		//enable or disable insurance, surrender, and ready buttons
		private void setInsuranceAndSurrenderState(boolean flag)
		{
			action_bar.button_insurance.setEnabled(flag);
			action_bar.button_surrender.setEnabled(flag);
			action_bar.button_ready.setEnabled(flag);
		}

		//enable or disable play buttons
		private void setPlayState(boolean flag)
		{
			action_bar.button_hit.setEnabled(flag);
			action_bar.button_double.setEnabled(flag);
			action_bar.button_stand.setEnabled(flag);
		}

		//enable or disable double button
		//this method is public because the game logic needs to access it
		public void setDoubleState(boolean flag)
		{
			action_bar.button_double.setEnabled(flag);
		}

		public void setBidFieldState(boolean flag)
		{
			HUD.fldBid.setEditable(flag);
		}

		//get the game mode from the drop down menu
		public int getGameMode()
		{
			return HUD.getModeComboBoxValue();
		}

		//enable or disable the drop down menu
		public void setModeComboBoxState(boolean flag)
		{
			HUD.cmbMode.setEnabled(flag);
		}

		//enable or disable the insurance button
		public void setInsuranceState(boolean flag)
		{
			action_bar.button_insurance.setEnabled(flag);
		}

		//return the amount entered into text field
		public int getBidAmount()
		{
			return Integer.valueOf(HUD.fldBid.getText());
		}

		public boolean validateBidAmount(int balance)
		{
			if(!Util.isNumeric(HUD.fldBid.getText()))
			{
				JOptionPane.showMessageDialog(this, "Bid must be a positive whole number", "Invalid Bid",
JOptionPane.INFORMATION_MESSAGE);
				return false;
			}
			int bid = getBidAmount();
			if(bid > balance)
			{
				JOptionPane.showMessageDialog(this, "Bid cannot be greater than the available balance", "Invalid Bid",
JOptionPane.INFORMATION_MESSAGE);
				return false;
			}
			else if(bid < 2)
			{
				JOptionPane.showMessageDialog(this, "Minimum bid is $2", "Invalid Bid",
JOptionPane.INFORMATION_MESSAGE);
				return false;
			}
			else if(bid % 2 != 0)
			{
				JOptionPane.showMessageDialog(this, "Bid must be even", "Invalid Bid",
JOptionPane.INFORMATION_MESSAGE);
				return false;
			}
			else	//valid bid
			{
				return true;
			}
		}

		//enable/disable buttons depending on game state
```

```
	public void setActionBarState(int state)
	{
		try {
			switch(state)
			{
			case STATE_DEAL:
				action_bar.button_deal.setEnabled(true);
				setInsuranceAndSurrenderState(false);
				setPlayState(false);
				action_bar.button_restart.setEnabled(false);
				break;
			case STATE_INSURANCE_OR_SURRENDER:
				action_bar.button_deal.setEnabled(false);
				setInsuranceAndSurrenderState(true);
				setPlayState(false);
				action_bar.button_restart.setEnabled(false);
				break;
			case STATE_PLAY:
				action_bar.button_deal.setEnabled(false);
				setInsuranceAndSurrenderState(false);
				setPlayState(true);
				action_bar.button_restart.setEnabled(false);
				break;
			case STATE_DEAL_AGAIN:
				action_bar.button_deal.setEnabled(true);
				setInsuranceAndSurrenderState(true);
				setPlayState(false);
				action_bar.button_restart.setEnabled(false);
				break;
			case STATE_RESULT:
				action_bar.button_deal.setEnabled(false);
				setInsuranceAndSurrenderState(false);
				setPlayState(false);
				action_bar.button_restart.setEnabled(true);
				break;
			default:
				throw new IllegalArgumentException();
			}
		}catch(Exception e){
			System.out.println("Invalid argument for setActionBarState");
		}
	}

	@Override
	public void actionPerformed(ActionEvent e)
	{
		Object source = e.getSource();
		if(source == action_bar.button_deal)
		{
			gameObj.onDeal();
		}
		else if(source == action_bar.button_hit)
		{
			gameObj.onHit();
		}
		else if(source == action_bar.button_double)
		{
			gameObj.onDouble();
		}
		else if(source == action_bar.button_stand)
		{
			gameObj.onStand();
		}
		else if(source == action_bar.button_insurance)
		{
			gameObj.onInsurance();
```

```
				}
				else if(source == action_bar.button_surrender)
				{
					gameObj.onSurrender();
				}
				else if(source == action_bar.button_ready)
				{
					gameObj.onReady();
				}
				else if(source == action_bar.button_restart)
				{
					gameObj.onRestart();
				}
		}
}
```

Util.java

```
package blackjack;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;

//utility class for commonly used operations
public class Util {

		//this method will always count an ace as 1, so the responsibility to account for it being an 11
		//falls to the game logic
		public static int calculateHandRank(ArrayList<Card> hand)
		{
				int rank = 0;
				for(Card card : hand)
				{
					rank += card.getRank().getNumericValue();
				}
				return rank;
		}

		//check if hand has an ace
		public static boolean hasAce(ArrayList<Card> hand)
		{
				for(Card card : hand)
				{
					if(card.getRank().equals(Rank.Ace)) return true;
				}
				return false;
		}

		//load image and return image object
		public static BufferedImage loadImage(final String path)
		{
				BufferedImage img = null;
				try {
					img = ImageIO.read(new File(path));
				} catch (IOException e) {
					// TODO Auto-generated catch block
					System.out.println("Failed to load " + path);
					e.printStackTrace();
				}
				return img;
		}
```

```
	//check if the string contains only numbers
	public static boolean isNumeric(String str)
	{
		if(str.matches("[0-9]+")) return true;
		return false;
	}
	//sleep milliseconds
	//used to simulate cool down for deal button because ActionListener can't handle release event
	public static void delay(long millis)
	{
		try {
			Thread.sleep(millis);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}
```

Main.java

```
package blackjack;

import javax.swing.SwingUtilities;

public class Main {
	public static void main(String[] args) {
		SwingUtilities.invokeLater(new Runnable() {
			@Override
			public void run() {
				new Game();
			}
		});
	}
}
```

References:

https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
https://www.w3schools.com/html/html_colors_rgb.asp https://www.geeksforgeeks.org/introduction-to-java-swing/ https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html
https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html
https://docs.oracle.com/javase/tutorial/2d/images/index.html
https://docs.oracle.com/javase/tutorial/2d/images/drawimage.html
https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html

https://www.geeksforgeeks.org/introduction-to-java-swing/

https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

https://mathbits.com/JavaBitsNotebook/GUIs/Introduction.html